



OWL Semantic Repository

ver. 2.8.0, 21 October 2005

System Documentation

1	Fact Sheet	2
2	Introduction	4
2.1	Reasoning strategies: Forward- vs. Backward-Chaining	5
2.2	IRRE Reasoning Schema	6
3	Persistence Strategy	7
4	Supported Semantics	8
5	Interfaces and RMI Access	12
6	Installation and Configuration	13
6.1	Distribution Contents	13
6.2	Running Standalone OWLIM for Remote Access	14
6.3	Configuration	14
7	Performance	17
7.1	City Benchmark	17
7.2	LUBM Benchmark	19
8	Implementation Notes	20
8.1	Read-only statements	20
9	References	21

1 Fact Sheet

OWLIM is a high-performance semantic repository, wrapped as a Storage and Inference Layer (SAIL) for the Sesame RDF database. OWLIM uses Ontotext's IRRE to perform OWL DLP reasoning, based on forward-chaining of entailment rules. The reasoning and query evaluation are performed in-memory. In the same time, a reliable persistence strategy assures data preservation, consistency and integrity. OWLIM can manage millions of explicit statements on desktop hardware. A limitation of OWLIM is the relatively slow delete operation. The upload and query evaluation are extremely fast even for huge ontologies and knowledge bases.

OWLIM development is partly supported by project SEKT, EU IST IP 2003-506826.

Availability and Contacts

- Version: 2.8.0, 21 Oct 2005.
- Download: <http://www.ontotext.com/owlim/v2.8/owlim.zip>
- Contact person: Damyan Ognyanov, damyan@sirma.bg, <http://www.ontotext.com/owlim>.

The Current Version Release Notes

- IRRE: OWLIM uses IRRE for in-memory reasoning and query evaluation. Sesame's standard in-memory SAIL, implementing the **RDFS****SchemaRepository**, is not used any longer.
- Upload and reasoning speed up: there is a major improvement to the upload and reasoning speed due to the usage of IRRE.
- Persistence: Persistence implementation get changed , but it is still compatible with previous releases of OWLIM in terms of storage formats and SAIL configuration options.
- Multi-threading: The inference process is not multi-threaded and even it is not thread-safe, so, it requires a special attention if used in a multi-thread context. OWLIM still uses multi-threading for other functions (e.g. persistence).
- Extended support for OWL: **intersectionOf**, **unionOf**, **AllDifferernt**, **someValuesFrom** are already handled in this version.

Interfaces, Standards, Requirements

- Nature: Java library without user interface.
- Interfaces (API, Web Services): Java API, RMI.
- Platform: JDK 1.4.2 and 1.5 (both 32-bit and 64-bit versions tested).

Supported standards:

- OWLIM is bound to the data and query standards supported by Sesame. The basic data standard is RDF, [9]. The query languages supported are: SeRQL, RQL, RDQL.
- **Syntaxes**: OWLIM uses N-Triples RDF syntax for persistence. Import and export for all the major RDF syntaxes are supported through Sesame: XML, N3, N-Triples.
- **Semantics**: OWLIM supports the DLP dialect, [5], of the OWL language, [4]. OWLIM supports also the standard model-theoretic semantics of RDFS, [7].

Required Libraries:

- Sesame (<http://www.openrdf.org>, [2]) is an open-source RDF database with support for RDF Inferencing and querying. OWLIM is a SAIL implementing **RDFS**SchemaRepository interface. OWLIM v2.8 is tested with Sesame releases 1.2.1 and 1.2.2.
- IRRE is an Inductive Reasoning Rule Engine allowing for forward-chaining and materialization with respect to entailment rules. IRRE v2.8 is preconfigured within OWLIM to support RDF(S) and OWL DLP.

Licensing

OWLIM License Agreement:

(c) Copyright 2005, Ontotext Lab, Sirma Group Corp.
135 Tsarigradsko Shosse, Sofia 1784, Bulgaria, <http://www.ontotext.com>.

This library is free software; you can redistribute it and/or modify it under the terms of the [GNU Lesser General Public License](#) (LGPL) as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version. This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details. You should have received a copy of the GNU Lesser General Public License along with this library; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA.

Licensing of Third Party Libraries:

- Sesame - (c) Copyright Aduna b.v. It is an open-source library, available under [LGPL](#).
- IRRE – (c) Copyright Ontotext Lab, Sirma Group Corp. IRRE is a proprietary non-open-source library. IRRE v2.8 is licensed free of charge for usage as an integral part of OWLIM v2.8. Re-distribution of IRRE in any form, different than the original OWLIM distribution package, is strictly forbidden. Any form of modification or reverse-engineering of IRRE is strictly forbidden. IRRE is distributed together with OWLIM, without any warranty.

Installation and Usage

OWLIM is distributed as a ZIP archive, which contains the **owlim-2.8.jar**, required libraries, configuration files, sources, documentation, and sample data. OWLIM is designed for usage as a Storage and Inference Layer (SAIL) of Sesame – there are no OWLIM specific interfaces. Sesame should be downloaded separately, it is not included in the OWLIM distribution.

Future Plans

- A more robust IRRE implementation that uses less memory is under development.
- Configurable index size to allow for fine-tuning of the SAIL for specific purposes.
- SPARQL support is under consideration.

2 Introduction

The semantic repositories allow for storage, querying, and management of structured data. They are designed as a replacement for the database management systems, offering easier integration of diverse data and more analytical power. In a nutshell, a semantic repository can dynamically interpret metadata schemata and ontologies. Compared to the approach taken in the relational DBMS, this allows for (i) easier changes and combinations of multiple data schemata and (ii) interpretation of the data. The later means that, for example, given a simple ontology, a semantic repository can return a UK mobile operator, when queried for telecom companies in Europe.

Over the last decade the Semantic Web¹ emerged as an area where the semantic repositories become equally important with the HTTP servers. This perspective boosted the development of a number of robust metadata and ontology standards (RDF(S) and OWL), under W3C driven community processes. Those play the role, which SQL had for the development and spread of the relational DBMS. Although designed for Semantic Web, these standards face increasing acceptance in areas like Enterprise Application Integration and life sciences. Sesame is one of the most popular semantic repositories with support for RDF(S), together with all the major syntaxes and query languages.

OWLIM is a high-performance semantic repository, which supporting the DLP fragment of OWL web ontology language, [4]. OWL DLP, [5], is a sub-language simpler than OWL Lite; it allows for compatibility across systems based on rules/logical programming (e.g. Datalog) and description logic (DL). Further, it is backward-compatible with RDFS, [7], with reasonable modelling constraints or transformations. More details on the semantics supported can be found in section 4.

OWLIM is wrapped as a Storage and Inference Layer (SAIL) for Sesame, named **OWLIMSchemaRepository**; it implements the **RdfSchemaRepository** interface. OWLIM uses Ontotext's IRRE (Inductive Reasoning Rule Engine) for reasoning and query answering.

The easiest way to use OWLIM is the so-called embedded mode, as a Java library. The distribution of OWLIM contains an RMI factory, named **CustomRMIFactory**, which allows access to the SAIL layer of a repository directly through RMI; more details are provided in section 5. The installation and configuration of OWLIM are discussed in section 6. More information related to various aspects of Sesame's specification, architecture, and implementations can be found in [2].

Although the reasoning is handled in-memory, OWLIM offers a robust persistency and backup strategy (presented in section 3), which assures data preservation, consistency, and integrity even in cases of abnormal termination. A separate thread is used to take care of the persistence to the temporary triple storage, in order to reduce the latency of OWLIM. Delete operations, when invoked in a transaction, just marks all the affected statements, while the actual job is done when the transaction is committed.

The efficiency of IRRE allows OWLIM to manage millions of explicit statements on desktop hardware. A limitation of OWLIM is the relatively slow delete operation. The upload and query

¹ <http://www.w3.org/2001/sw/>

evaluation are extremely fast even for huge ontologies and knowledge bases, as demonstrated through a number of benchmarks, presented in section 7.

2.1 Reasoning strategies: Forward- vs. Backward-Chaining

Here we provide an oversimplified introduction on some basic choices related to the implementation of repositories with inference functionality. Two principle strategies can be outlined, as follows:

- **Forward chaining:** to start from the known facts and to perform inference in an inductive fashion. The goals of such a way of reasoning could be different: to compute the *inferred closure*²; to answer a particular query; to infer a particular sort of knowledge (e.g. the class taxonomy).
- **Backward chaining:** to start from a particular fact or a query and to verify it or get all possible results, using deductive reasoning. In such case, essentially, the reasoner decomposes or otherwise transforms the query or the fact into alternative or simpler facts, which are available in the knowledge base or can be proven through further, recursive, transformations.

These strategies have different strong and weak points, which are studied quite well in the history of knowledge representation and expert systems. Hybrid strategies (involving partial forward- and backward-chaining) are also possible and proven to be efficient in many contexts.

Let us imagine a repository which performs total forward-chaining, i.e. it tries to make sure that, after each update to the KB, the inferred closure is computed and made available for query evaluation or retrieval. This strategy is known as *materialization*. Suppose also that the repository is reasoning over a monotonic logic³, which means that each time when new explicit facts (or statements) are added to the KB, new fact can extend the inferred closure, but in no case old facts should be removed. Let us call such an inference strategy, taken together with monotonic assumption, *total materialization*, to avoid ambiguity with various partial materialization approaches. The principle advantages and disadvantages of this schema can be summarized as follows:

- Upload/store/addition of new facts is relatively slow, because the repository is trying to extend the inferred closure after each transaction for modification;
- Deletion of facts is also slow, because the repository should remove from the inferred closure all the facts which are not true any longer.
- The maintenance of the inferred closure usually requires considerable additional space (RAM, disk, or both, depending on the implementation);
- Query and retrieval are fast, because no deduction, satisfiability checking, or other sorts of reasoning are required. The evaluation of the queries becomes computationally comparable to the same task for relation database management systems (RDBMS).

² The term *inferred closure* is defined here as follows: the extension of a KB (or a graph of RDF triples, in the case of RDF/OWL repository) with all the facts (implicit triples), which could be inferred from it, based on the semantics enforced.

³ A discussion between Ian Horrocks, Pat Hayes, and Seth Russell on the relevance/utility of monotonic logics for the Semantic Web can be found at <http://lists.w3.org/Archives/Public/www-rdf-logic/2001Jul/0064>

Probably the most important advantage of inductive systems, based on total materialization, is that those can easily benefit from RDBMS-like query optimization techniques, as long as all the data is available at query time. The latter makes possible for the query evaluation engine to use statistics and other means in order to make “educated” guesses about the cost and result of a particular constraint. These optimizations are much more complex in the case of deductive query evaluation.

Total materialization is adapted as a reasoning strategy in a number of the popular Semantic Web repositories, including many of the standard configurations of Sesame and Jena⁴.

2.2 IRRE Reasoning Schema

Ontotext's IRRE (Inductive Reasoning Rule Engine) performs reasoning, based on forward-chaining of entailment rules. Its reasoning strategy is total materialization, as introduced in the previous section.

The implementation of IRRE relies on a compile stage, during which the entailment rules are compiled into chunks of Java code that are post-processed automatically and merged together to generate the main entry point for the inferencer. The reasoning and query evaluation are performed in-memory; the latter means that the full content of the repository is loaded and maintained in a proprietary representation in the main memory, which allows for efficient retrieval and query answering.

IRRE can be configured with a set of rules, which determines the supported semantic. Each rule has a set of premises, which conjunctively define the body of the rule. The premises are RDF statements, which can contain any number of variables. The head of the rule is one or more consequences, each of which is again an RDF statement. The consequences can contain free variables, i.e. such which are not used in the body of the rule. In such cases, those are bound to a newly created anonymous nodes. The capability of having multiple consequences in the head is allowed. The basic language supported is a sub-set of the Horn logic.

⁴ <http://www.hpl.hp.com/semweb/jena.htm>

3 Persistence Strategy

The persistency of OWLIM is implemented through N-Triples files; it is based on the same strategy employed into old version of **RdfSchemaRepositoryV2** SAIL of Sesame, developed by Ontotext Lab back in 2004. The repository can be configured to spread into several files. There is one of these files (let us name it **persist**) that is considered both an input for loading and a target for modifications: new statements can be stored there and existing ones can be deleted. All other files are considered read-only. This allows implementation of a scenario with multiple static bodies of background knowledge and data (ontologies, knowledge bases, etc) and single file which corresponds to the dynamic content of the repository. Such a setup is useful in cases of layered and modularized ontologies or KB. The configuration of the repository is discussed in section 6.

The backup strategy implemented ensures that no loss of newly asserted triples can occur in cases of power failure or abnormal termination. It works in the following way:

- All new explicit triples are persisted into an auxiliary file (specified with the **new-triples-file** parameter in the configuration file), when the write buffer gets full or when or when the **commitTransaction()** method is invoked. The size of this specific write buffer still cannot be modified in current version of OWLIM; the default size is 20,000 statements.
- The synchronization of the in-memory contents of the repository and its **persist** file can happen either on a regular basis (after a timeout configured via the **syncDelay** parameter), or when OWLIM is shutdown properly.
- In case that a 'regular' synchronization completes, the **new-triples-file** is renamed to an file with a **~bak** extension and then a new empty one is created in its place.
- In case of abnormal process termination, the contents of the **new-triples-file** are added to the repository the next time the SAIL is initialized, right after the processing of the **persist** file is complete.

Although relatively simple, this strategy had proven to be very efficient and reliable, through the couple of years for which **RdfSchemaRepositoryV2** and OWLIM has been used as a semantic repository for different applications of the KIM semantic annotation, indexing, and retrieval platform, <http://www.ontotext.com/kim>.

4 Supported Semantics

Within OWLIM, IRRE is configured with a set of rules, which cover the model theoretic semantics of RDFS, as defined in [7]. OWLIM extends these rules with a set, which support the following OWL constructs: **SymmetricProperty**, **TransitiveProperty**, **inverseOf**, **equivalentClass**, **equivalentProperty**, **sameAs**, **FunctionalProperty**, **InverseFunctionalProperty**, **onProperty**, **allValuesFrom**, **someValuesFrom**, **hasValue**, **unionOf**, **intersectionOf**, **differentFrom** and **AllDifferent**. The major limitations to the support of those primitives are as follows:

- **onProperty** is supported only in the context of **allValuesFrom**, **someValuesFrom**, **hasValue**;
- **allValuesFrom**, **someValuesFrom** and **unionOf** are limited so that the support includes rules that close the semantics only in one direction. This keeps the inference monotonic, which means that such complete class definitions are not used to infer a class membership in the opposite direction;
- **differentFrom** statements are only generated from **AllDifferent**; no sort of inference is carried out on top of them. It is left to the application to check inconsistency in cases when a couple of RDF nodes linked though both **sameAs** and **differentFrom** properties.

In terms of OWL compliance, OWLIM supports a sub-set of OWL DLP as defined in [5], which is a proper sub-set of OWL Lite. Extensions towards OWL DHL are possible through extension of the set of rules given to IRRE.

The entailment rules, used in OWLIM to support the semantics of the above primitives are listed below. The rules supporting the RDFS semantics are not listed here; one can find them in section 7 of [7].

1. Support for **owl:SymmetricProperty**

```
prop <rdf:type> <owl:SymmetricProperty> &&
xxx prop yyy
-->
yyy prop xxx
```

2. Support for **owl:TransitiveProperty**

```
xxx prop yyy &&
yyy prop zzz &&
prop <rdf:type> <owl:TransitiveProperty>
-->
xxx prop zzz
```

3. Support for **owl:inverseOf**

```
xxx prop yyy &&
prop <owl:inverseOf> invprop
-->
yyy invprop xxx
```

- Support for **owl:sameAs**. Implemented through replication of the statements where the equivalent resources appear as subject, predicate, or object.

```
xxx <owl:sameAs> yyy &&
xxx pred zzz
-->
yyy pred zzz

pred <owl:sameAs> other &&
xxx pred yyy
-->
xxx other yyy

xxx <owl:sameAs> yyy &&
zzz pred xxx
-->
zzz pred yyy
```

- Support for **owl:FunctionalProperty**. Implemented as equality (**owl:sameAs**) of multiple objects for one and the same subject in statements where a functional property is used as a predicate.

```
xxx pred yyy &&
pred <rdf:type> <owl:FunctionalProperty> &&
xxx pred zzz
-->
yyy <owl:sameAs> zzz
```

- Support for **owl:InverseFunctionalProperty**. Analogous implementation to the one for **owl:FunctionalProperty**.

```
xxx pred yyy &&
pred <rdf:type> <owl:InverseFunctionalProperty> &&
zzz pred yyy
-->
xxx <owl:sameAs> zzz
```

- Support for restrictions **owl:onProperty** of type **owl:allValuesFrom**. The support is limited only to infer a class membership for the values of the triples, which subjects are members of the restriction and do not infer a restriction membership in the opposite direction of the rule.

```
restriction <owl:allValuesFrom> yyy &&
restriction <owl:onProperty> ppp &&
uuu ppp vvv &&
uuu <rdf:type> restriction
-->
vvv <rdf:type> yyy
```

- Support for restrictions **owl:onProperty** of type **owl:hasValue**. The support includes inferring a new triple with the defined predicate and value for the nodes that are members of the restriction. Further, OWLIM derives a **Restriction** membership for nodes involved as subjects in statements, where the predicate and the object match those specified in the restriction.

```
restriction <owl:hasValue> yyy &&
restriction <owl:onProperty> ppp &&
uuu <rdf:type> restriction
-->
uuu ppp yyy
```

```
instance property value
restriction <owl:onProperty> property
restriction <owl:hasValue> value
-->
instance <rdf:type> restriction
```

9. Support for restrictions **owl:onProperty** of type **owl:someValuesFrom**. The support is limited only to infer a restriction membership for nodes which are related to other nodes (values) of the corresponding class through the restricted property.

```
value <rdf:type> class
restriction <owl:onProperty> property
restriction <owl:someValuesFrom> class
instance property value
-->
instance <rdf:type> restriction
```

10. The support for property **owl:equivalentClass** is implemented as it is declared to be symmetric and transitive and also a sub-property of **rdfs:subClassOf**. This approach was chosen for performance optimization, but the users should be aware that “side effects” while traversing the class hierarchy are possible. It also derives class equivalence for classes that are mutual subclasses of each other using the next rule:

```
x <rdfs:subClassOf> y
y <rdfs:subClassOf> x
-->
x <owl:equivalentClass> y
```

11. The support for property **owl:equivalentProperty** is implemented as it is declared to be a sub-property of **rdfs:subPropertyOf**.

12. The support for **owl:intersectionOf** relies on the following four rules that infer class membership in both directions. First, for the explicit members of the intersection, they derive a class membership for each of the intersected classes and second, for each instance, member of all the intersected classes, it derives a class membership to the intersection as well.

```
c <owl:intersectionOf> x
x <rdf:first> y
-->
c <rdfs:subClassOf> y

c <owl:intersectionOf> x
x <rdf:rest> y
y <rdf:first> z
-->
c <rdfs:subClassOf> w
w <owl:intersectionOf> y

i <rdf:type> c
n <rdf:first> c
n <rdf:rest> l
a <owl:intersectionOf> l
b <owl:intersectionOf> n
i <rdf:type> a
-->
i <rdf:type> b

a <owl:intersectionOf> l
l <rdf:first> c
```

```

l <rdf:rest> <rdf:nil>
-->
c <rdfs:subClassOf> a

```

13. The support for **owl:AllDifferent** is implemented with a pair of rules that entails **owl:differentFrom** statements for the distinct members of the collection by which it is defined.

```

x <owl:distinctMembers> m
m <rdf:rest> n
-->
x <owl:distinctMembers> n

x <owl:distinctMembers> m
x <owl:distinctMembers> n
m <rdf:first> i
n <rdf:first> j
-->
i <owl:differentFrom> j

```

14. The support for **owl:unionOf** is limited to deriving a class membership to the classes that form the union, but only for the explicit members of the union:

```

a <owl:unionOf> m
m <rdf:rest> n
-->
z <owl:unionOf> n
z <rdfs:subClassOf> a

a <owl:unionOf> m
m <rdf:first> c
-->
c <rdfs:subClassOf> a

```

Note: Alongside the above OWL-specific rules, below follow a few which are specific for the PROTON ontology, [12], and its usage within KIM. At present, this is an ad-hoc solution, which should not affect scenarios where the corresponding PROTON properties are not used. It is planned that such extensions will be handled through a more standard rule mechanism in the future releases.

15. Support for property **protons:transitiveOver**, with the aid of which, one can define relationships like between the **rdf:type** and **rdfs:subClassOf**. The namespace prefix **protons** is defined as <http://proton.semanticweb.org/2005/04/protons>.

```

xxx prop yyy &&
prop <protons:transitiveOver> other &&
yyy other zzz
-->
xxx prop zzz

```

16. The following KIM-specific rule is also added. IMO, it hardly can interfere with the inference used by the regular SAIL user. The namespace prefix **protont** is defined as <http://proton.semanticweb.org/2005/04/protont>.

```

xxx <protont:roleHolder> yyy &&
xxx <protont:roleIn> zzz &&
yyy <rdf:type> <protont:Agent>
-->
yyy <protont:involvedIn> zzz

```

5 Interfaces and RMI Access

OWLIM can be used through the standard interfaces of Sesame. Documentation of these interfaces (including Javadoc) can be found at <http://www.openrdf.org>, see [11]. In a nutshell, an application can use Sesame in two ways:

- Higher level interfaces which allow for generic operation and query answering;
- Through the SAIL interfaces, which allow low-level access to a triple repository.

The usage of the new custom RMI factory can be enabled through specifying it explicitly into a **rmi-factory** tag of the **system.conf**. Example:

```
<rmi-factory enabled="true"
  class="com.ontotext.util.rmi.CustomRMIFactory"port="1099"/>
```

It will act as the default factory, but instead of returning instances that only implement **SesameRepository**, it will return instances implementing the **com.ontotext.util.rmi.SailAccessor** interface, too. Here follows a fragment of code illustrating how to access the top SAIL interface of a repository via RMI in case the **CustomRMIFactory** is enabled server-side:

```
...
import com.ontotext.util.rmi SailAccessor;
...
SesameService ss = Sesame.getService(
new URI("rmi://localhost:1098"));
ss.login("admin", "REPLACE_ME");
SesameRepository sr = ss.getRepository("kim");
Sail topSail = null;
if (sr instanceof SailAccessor)
topSail = ((SailAccessor)sr).getSail();
```

6 Installation and Configuration

OWLIM is a specific plug-in for Sesame; configuring, running, and using OWLIM to a major degree means, doing it for a specific configuration of Sesame. Please refer to the Sesame's online documentation [11].

OWLIM can be used from an application in two modalities:

- **Embedded mode:** as a library, invoked in the same process as the application, which is using it.
- **Remote access:** running as a standalone server in a separate process (possibly on a different machine); in this case the applications communicate with it through RMI calls.

In both cases, the OWLIM jar file (Java library) should be included in the Java class-path of the application. This file can be found in the **lib** subfolder of this distribution.

NOTE: The OWLIM jar must be placed before **sesame.jar** in the class-path of the application.

Instructions and samples on using Sesame from an application in embedded mode can be found in its system documentation, [11]. In a nutshell, one should obtain a **SesameService** instance, than a repository, as demonstrated in the code snippet below:

```
// Example: initialize, using an external configuration file:
File sysConf = new File("./sesame.conf");
LocalService ss = Sesame.getService(sysConf);

ss.login("admin", "<REPLACE_ME>");

// get a local repository
LocalRepository rep = (LocalRepository)ss.getRepository("owlim");

// issue a query
QueryResultsTable result =
    rep.performTableQuery(QueryLanguage.SERQL,
        "select * from {X} rdf:type {rdfs:Class}");
// just dump the results
for (int i = 0; i < result.getRowCount(); i++) {
    System.out.println(result.getValue(i, 0));
}

// shutdown the local service
ss.shutdown();
```

Using OWLIM through RMI is almost transparent – the small differences are in the way you get the **SesameService** instance in the beginning, as it is explained in section 5.

6.1 Distribution Contents

The distribution of OWLIM includes:

- **lib** folder: contains OWLIM as a JAR file (Java library), together with the all the necessary third party libraries (except those of Sesame).
- **doc** folder: system documentation and test documentation;
- **src** folder: the Java sources of OWLIM and the RMI factory;

- **test** folder: Benchmarks and unit tests; contains sources, data and documentation, documented in [10]. The two major applications are the LUBM benchmark and JUnit tests implementing checks for the normative Positive and Negative-entitlement OWL test cases, [3]. Those can also be used as a sample applications;

6.2 Running Standalone OWLIM for Remote Access

In summary, one should start a Sesame service and bind it into RMI registry, before to try accessing it form the application. In order to allow Sesame to close properly, one should shutdown the service properly. Although OWLIM can survive abnormal termination of the process without inconsistencies in the repository, we recommend that the proper Sesame shutdown mechanism is used (this affects the initialization time for the next run).

The distribution of OWLIM contains a sample configuration and a (Windows) **cmd** script, which allow start-up and shutdown of OWLIM. A Sesame distribution should also be obtained and installed beforehand. The concrete steps are as follows:

1. Make sure that Java 1.4.2 or later is installed;
2. Download Sesame v1.2.2 from <http://www.openrdf.org>;
3. Unpack Sesame at a convenient place and modify the **owlim_control.cmd**, so that the **SESAME_LIB** environment variable points to the folder containing the jar files from the Sesame distribution;
4. To start it, type "**owlim_control.cmd start**" and, optionally, provide it with the pathname of the configuration file and the number of the TCP/IP port to bind it to. The default configuration file is located at relative path **./test/test/sesame.conf** in the distribution. The default port is 1098.

OWLIM-configured Sesame will be started in a standalone mode. It can be shutdown with the aid of the "**owlim_control.cmd stop <portNo>**" shell command, where the port number is again optional.

The sample configuration file loads the PROTON ontology, [12], in the default repository "**owlim**" as background knowledge and uses **./kb/kb.nt** file as **persist** file for that repository. The configuration file is provided just as a sample; the user is expected to modify it, e.g. by providing its own ontologies and data as a "background knowledge", if such exist. To begin with a clear repository, one should remove the PROTON specific parts from the configuration file, declared in **imports** and **defaultNS** parameters of the "owlim" repository definition. Please, follow the next section to get familiar with the basics on how to configure Sesame to use OWLIM.

6.3 Configuration

Sesame is configured through an XML configuration file, which can be provided in the stage when retrieving of the **SesameService** instance. In case of remote usage, Sesame has to be configured at the server side. Follow the instructions on how to setup the related parameters, in the configuration file.

The SAIL itself can be initialized with a set of schema files; it should also include the original OWL XML/RDFS file (**owl.rdfs**) – a copy of it is included in this distribution for the sake of convenience.

Here is a list of the available configuration parameters and their description. To specify a .NT file, in which to persist the repository contents:

```
<param name="file" value="./kb/kb.nt" />
```

To specify the serialization format for the main persist file

```
<param name="dataFormat" value="ntriples" />
```

To specify whether to use a compression on that file:

```
<param="compressFile" value="no"/>
```

The **new-triples-file** specified the name of a file, where OWLIM stores temporarily the new triples that have just been added to the repository. It is used in case of abnormal termination, so its contents will be automatically added to the repository right after the initialization with the main persist file (see **param="file"**) completes. If the SAIL has been shutdown properly or successfully synchronized with the main **persist** file, then the contents of this **new-triples-file** are not necessary since each triple, mentioned there, will already be a part of the main **persist** file. If this parameter is omitted in the SAIL configuration, the backup strategy is switched off and the repository contents will be persisted only when the SAIL is shutdown properly, which happen when one invoke the repository's **shutdown()** method.

```
<param="new-triples-file" value="./kb/new-temp-triples.nt" />
```

The **base-URL** parameter specifies the default namespace for the persist file. We recommend the usage of a non-empty namespace, because it is used to assure the uniqueness of the anonymous nodes that may appear in the repository.

```
<param="base-URL" value="http://www.ontotext.com/kim/2004/12/wkb#" />
```

The **imports** parameter allows specification of a list of schema files to include – all statements, found in these files, are treated as read-only and cannot be removed. The serialization format is assumed to be RDFS/XML, unless the file has a **.NT** extension.

```
<param name="imports" value="list of ontology files delimited by ';' " />
```

In conjunction with the above parameter, use the **defaultNS** parameter to specify the default namespaces for each of the imported files. The number of ';' semicolon-delimited tokens in both parameters should be the same.

```
<param name="defaultNS" value="namespaces list for the imported files" />
```

Here follows an example **<repository>** section that may appear in the **system.conf** file of Sesame:

```
<repository id="owlim">
<title>OWLIM with PROTON ontology</title>
<sailstack>
<sail class="org.openrdf.sesame.sailimpl.memory.OWLMemSchemaRepository">
<param name="file" value="./kb/kb.nt" />
<param name="compressFile" value="no"/>
<param name="dataFormat" value="ntriples" />
<param name="new-triples-file" value="./kb/new-temp-triples.nt" />
<param name="syncDelay" value="-1"/>
```

```

<param name="dropOnRemove" value="false" />
<!-- semicolon should be used as delimiter -->
<param name="imports" value=
"/ontology/owl.rdfs;
./ontology/protons.owl;
./ontology/protont.owl;
./ontology/protonu.owl;
./ontology/protonkm.owl;
./ontology/kimso.owl;
./ontology/kimlo.owl"/>
<param name="defaultNS" value=
"http://www.w3.org/2002/07/owl#;
http://proton.semanticweb.org/2005/04/protons#;
http://proton.semanticweb.org/2005/04/protont#;
http://proton.semanticweb.org/2005/04/protonu#;
http://proton.semanticweb.org/2005/04/protonkm#;
http://www.ontotext.com/kim/2005/04/kimso#;
http://www.ontotext.com/kim/2005/04/kimlo#" />
</sail>
</sailstack>
<!--Access Control List can contain zero or more 'user' elements-->
<acl worldReadable="false" worldWritable="false">
<user login="admin" readAccess="true" writeAccess="true"/>
</acl>
</repository>

```

7 Performance

There are several concerns, which normally arise for reasoners implemented after the approach taken in IRRE, and thus OWLIM – see sections 2.1 and 2.2. Total materialization performed in-memory puts on question the upload speed for big repositories, the scale (limited by the RAM available), and the speed of the delete operation. A number of tests and evaluations, had been performed in order to evaluate the scalability and performance of OWLIM. Below we present a couple of them, proving that OWLIM scales up to millions of statements even on commodity desktop hardware. Based on the limited evaluation results, publicly available, OWLIM is the fastest OWL repository currently available!

7.1 City Benchmark

To explore the scalability of OWLIM, we conducted a benchmark experiment on several systems with different hardware specifications, as presented on Table 1.

Table 1. Hardware and Software Configurations for the Different Runs

Name	Configuration	RAM	JDK
2Opt7600m	2xOpteron 2.0GHz, Win 2003 64-bit	7600mb, DDR400	JDK 1.5 64-bit
2Opt2600m	2xOpteron 2.4GHz, Red Hat Linux v.3	2600mb, DDR333	JDK 1.5 64-bit
2Xeo1600m	2xXeon 2.4GHz, Win XP	1600mb, DDR333	JDK 1.5 32-bit
Piv800m	Pentium IV, 3.0GHz, Win XP	800mb, DDRII 533	JDK 1.5 32-bit
PM680m	Pentium Mobile 1.6GHz, Win XP	680mb, DDR266	JDK 1.5 32-bit

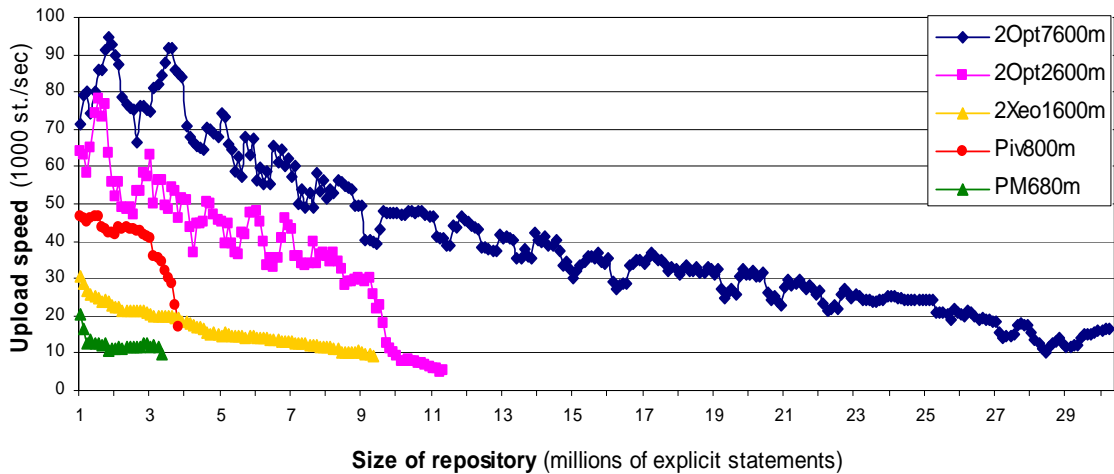
The experiment can be summarized as follows:

- initially the repository is populated with about 500k explicit statements – a real ontology (PROTON, [12]) and a knowledge base – the small version of KIM's WKB (World Knowledge Base), <http://www.ontotext.com/kim>;
- the repository is being extended with synthetic descriptions of cities, each transaction adding one city described in about 10k statements. The cities are linked to real provinces (randomly chosen from the WKB). Ten synthetic organizations are created and "located" into each city. Each organization is described to be active in an industry sector – again randomly chosen from the WKB. Finally, 38 persons are created and defined to have positions within each of the organizations. This way WKB is extended with LDAP like data in realistic manner.
- A couple of test queries (in SeRQL) are evaluated after the addition of each 10 cities. The second one (Q2, results presented below), was more complex; it is composed of pattern of 12 statement-joins and LIKE "*xyz*" literal constraint.
- Delete is also tested on each 100 cities generated. The test goes until a certain RAM limit is reached.

The results can be summarized as follows:

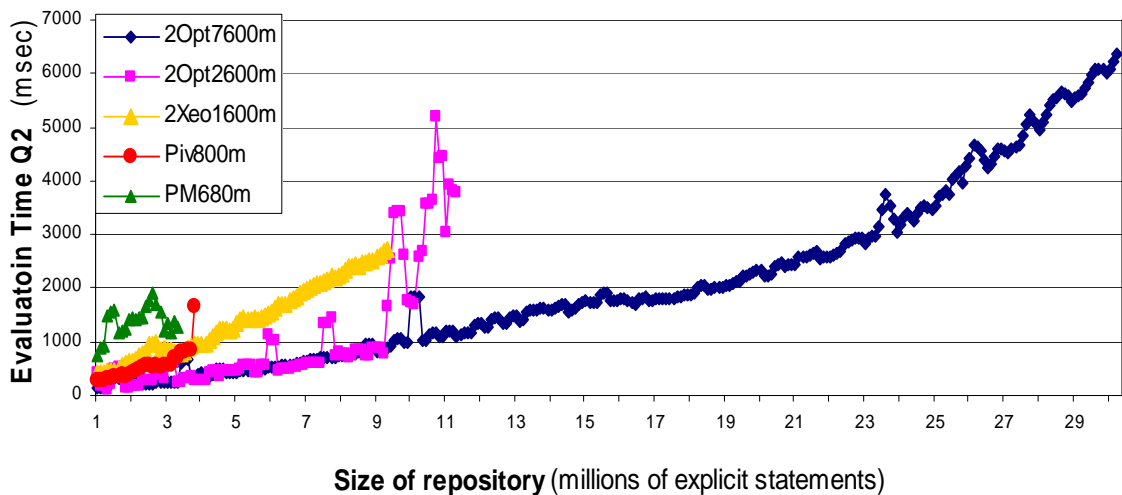
- The upload speed (including inference and storage) varies in the range of 10,000-100,000 statements/second, depending on the machine and the size of the repository – see Figure. 1;

Figure. 1. Upload Speed



- The maximum size of the repository varies from couple million statements on a notebook to 30 million statements on a server with 8GB of RAM. As it can be expected, the 64-bit Java virtual machine requires slightly more memory for the same size of the repository.
- The time for query evaluation grows linearly with the size of the repository and the result count. It starts at tens of milliseconds, when the repository contains few millions of statements, and grows to few seconds when the repository gets bigger. Because all the query results are fetched, the total time for the query is affected by the size of the result, which grows linearly with the size of the repository, to rich tens of thousands of results – see Figure. 2.

Figure. 2. Query Evaluation and Result Fetching Time



- The delete operation is relatively slow, as it can be expected due to the straight forward invalidation of the inferred closure - the time for finalization of a delete transaction on machine Piv800m varies from with respect to the size of the repository as follows: 25 sec. for 1.5M st.; 44 sec. for 2.5M st.; 63 sec. for 3.5M st. In other words, it grows linearly with about 20 sec. for each new million of statements in the repository.

The results of the evaluation of a previous version of OWLIM (v.2.0) are published in [8], together with a more comprehensive description of the City benchmark platform.

7.2 LUBM Benchmark

The Lehigh University evaluation, [6], is one of the most comprehensive benchmark experiments published recently. It evaluates the upload and query performance of 4 systems: memory-based Sesame, database-based Sesame, DLDB-OWL, and OWLJesKB. The benchmark was made with a relatively simple ontology about the organizational structure of a university and with synthetically generated datasets – for each university, a number of departments and employees with descriptions and relations among them were generated. The performance of the systems was measured in the course of incrementally increasing the number of the universities (from 1 to 50). The smallest set is 8MB and the largest one (for 50 universities) is 583MB, described in 1000 owl files, with a total of about 6 million statements.

LUBM benchmarks is equipped with sample data, java interfaces and programs which allow its adoption for different repositories. An adapter of OWLIM for LUBM is provided together with the OWLIM distribution and documented in [10]. The results of the LUBM evaluation of OWLIM on a desktop machine (configuration Piv800m, but given 900MB of RAM to the Java machine) can be summarized as follows:

- OWLIM loads LUBM(50,0) in 417 sec, i.e. about 7 min. According to the results published in [6], the only other system, which can load it, does so in more than 12 hours!
- All the 14 queries are answered correctly. The query evaluation times for LUBM (50,0) are given in Table 2.

Table 2. LUBM(50,0) Query Evaluation Time for OWLIM

Query	Time (ms)	Result #	Query	Time (ms)	Result #	Query	Time (ms)	Result #
1	531	4	6	7,599	519,842	11	10	224
2	1,656	130	7	88	67	12	93	15
3	10	6	8	6,838	7,790	13	10	228
4	5	34	9	12,297	13,639	14	1,344	393,730
5	5	719	10	31	4			

8 Implementation Notes

This section contains a number of implementation details and specific issues, which can appear interesting for better understanding of the how OWLIM works and troubleshooting.

8.1 Read-only statements

The explicit statements, which are imported from the “background knowledge” files (that were described in the SAIL configuration using `import` parameter), are treated as 'read-only' and cannot be removed. This prevents the applications from editing ontologies and schemata, which could be outside their authority. All other explicit statements (those imported from the `persist` file or added programmatically), can be manipulated at will.

The axiomatic statements (i.e. such that could be inferred from an empty repository, but had never been formally asserted) are also treated as read-only for consistency reasons; this corresponds to the standard Sesame behavior.

9 References

- [1] Beckett, D. (editor). (2004). *RDF/XML Syntax Specification (Revised)*. W3C Recommendation 10 Feb. 2004. <http://www.w3.org/TR/rdf-syntax-grammar/>
- [2] Broekstra, J. (2005). *Storage, Querying and Inferencing for Semantic Web Languages*. Ph.D. Thesis, Vrije Universiteit Amsterdam, SIKS Dissertation Series No. 2005-09, ISBN 90 9019 2360. <http://www.cs.vu.nl/~jbroeks/#pub>
- [3] Carroll, J. J; De Roo, Jos. (2004). *OWL Web Ontology Language: Test Cases*. W3C Recommendation 10 Feb. 2004. <http://www.w3.org/TR/owl-test/>
- [4] Dean, M; Schreiber, G. – editors; Bechhofer, S; van Harmelen, F; Hendler, J; Horrocks, I.; McGuinness, D. L; Patel-Schneider, P. F.; Stein, L. A. (2004). *OWL Web Ontology Language Reference*. W3C Recommendation, 10 Feb. 2004. <http://www.w3.org/TR/owl-ref/>
- [5] Grosz, B; Horrocks, I; Volz, R; Decker, St. (2003). *Description Logic Programs: Combining Logic Programs with Description Logic*. In Proc. of WWW2003, Budapest, May 2003.
- [6] Guo, Y; Pan, Z; and Heflin, J. (2004). *An Evaluation of Knowledge Base Systems for Large OWL Datasets*. Journal of Web Semantics, 3(2), 2005, pp158-182. <http://www.websemanticsjournal.org/ps/pub/2005-16>
- [7] Hayes, P. (2004). *RDF Semantics*. W3C Recommendation 10 Feb. 2004. <http://www.w3.org/TR/2004/REC-rdf-mt-20040210/>
- [8] Kiryakov, A; Ognyanov, D; Manov, D. (2005). *OWLIM – a Pragmatic Semantic Repository for OWL*. In Proc. of International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS 2005), WISE 2005, 20 Nov, New York City, USA.
- [9] Klyne, G; Carrol, J. J; (eds). (2004). *Resource Description Framework (RDF): Concepts and Abstract Syntax*. W3C Recommendation 10 Feb. 2004. <http://www.w3.org/TR/rdf-concepts/>
- [10] Ontotext Lab. (2005). *OWLIM Tests and Benchmarks*. v2.8.0, DRAFT. <http://www.ontotext.com/owlim/v2.8/OWLIMTests.pdf>
- [11] Sesame Online Documentation. <http://www.openrdf.org/doc/sesame/users/index.html>
- [12] Terziev, Iv; Kiryakov At; Manov, D. (2005) *Base upper-level ontology (BULO) Guidance*. Deliverable D1.8.1 of the SEKT Project. http://www.ontotext.com/sekt/D1_8_1.pdf